Documentation for Predicting Optimal Communication Time Using Historical Data (Team- umiam)

1.Introduction

In this challenge, we aim to enhance customer engagement by predicting the optimal time slots for sending marketing emails. The bank needs to optimize communication frequency due to cost and regulatory limitations, ensuring that emails are sent when customers are most likely to open them.

Our goal is to develop a machine learning model that predicts a personalized time slot ranking for each customer, maximizing the likelihood of engagement.

2. Data Preprocessing:

2.1. Handling Missing Values

Managing missing data is crucial to maintaining the dataset's integrity. Columns with more than 50% missing values are identified and removed, as these are unlikely to provide meaningful insights. For the remaining columns, missing values are handled based on their data type. Numerical columns are imputed with their median to minimize the effect of outliers, while categorical columns are filled with their most frequent value (mode) to retain common patterns. This step uses a progress bar, powered by tqdm, to visualize the progress of filling missing values, ensuring that the dataset has no critical gaps that could impact analysis or modeling.

2.2. Data Type Corrections

To ensure the dataset is prepared for numerical and categorical operations, data type corrections are performed. Columns with object data types are checked for possible conversion to numerical values using pd.to_numeric(). If conversion is successful without introducing new missing values, the column is retained as numerical. Otherwise, columns are converted to categorical data types. This process helps reduce ambiguity in the dataset and ensures that all features are correctly typed for analysis or modeling.

2.3. Removing Low-Variance Features

Features with low variance provide little to no discriminatory power and can be safely removed to simplify the dataset. Variance is calculated for each numerical column, and those with a variance below a specified threshold (e.g., 0.01) are dropped. This step reduces the dimensionality of the

dataset by removing redundant features, thus improving the efficiency of downstream tasks such as machine learning model training.

2.4. Removing Highly Correlated Features

Highly correlated features can cause redundancy and multicollinearity, which can negatively impact model performance. A correlation matrix is calculated for the numerical features, and pairs of columns with a correlation coefficient above a certain threshold (e.g., 0.9) are identified. From each pair, one feature is removed to retain the most informative feature set. This step ensures that the dataset remains concise and free from unnecessary duplication of information.

2.5. Scaling Numerical Features

To normalize numerical columns for consistent analysis, scaling is applied using the MinMaxScaler. First, numerical columns that are not binary are identified. These columns are then scaled to a range of [0, 1], ensuring that all features have comparable magnitudes. Scaling is essential for models that are sensitive to the scale of input data, such as distance-based algorithms or gradient-based optimizers.

2.6. Aggregation

To summarize the dataset and make it more compact, rows are aggregated based on a grouping column, typically customer_code. For each group, numerical columns are aggregated by their mean, categorical columns by their mode, and datetime columns by their median. This approach ensures that the aggregated data represents the group's central tendency. The process also allows for a more manageable dataset size, making analysis and modeling more efficient while retaining key insights.

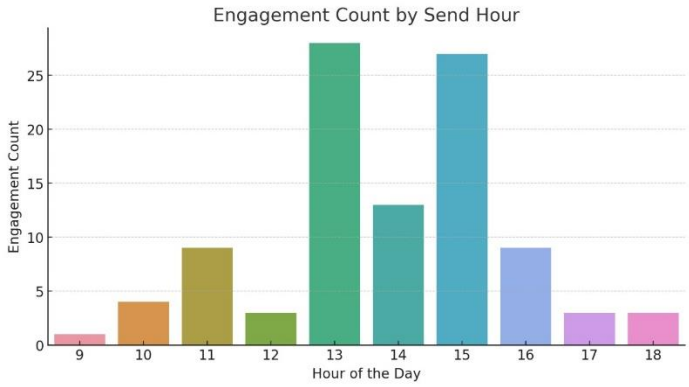
2.7. Merging Datasets

Merging multiple datasets into a single cohesive dataset is an essential step in integrating various data sources. In this workflow, datasets are merged using an outer join based on the customer_code column. This approach ensures that all rows from both datasets are included, even if some keys are missing in one of the datasets. Before merging, rows with missing customer_code values are removed to avoid inconsistencies. The progress of the merge operation is tracked using tqdm. This step creates a unified dataset that combines all relevant information from different sources.

2.8. Feature Engineering

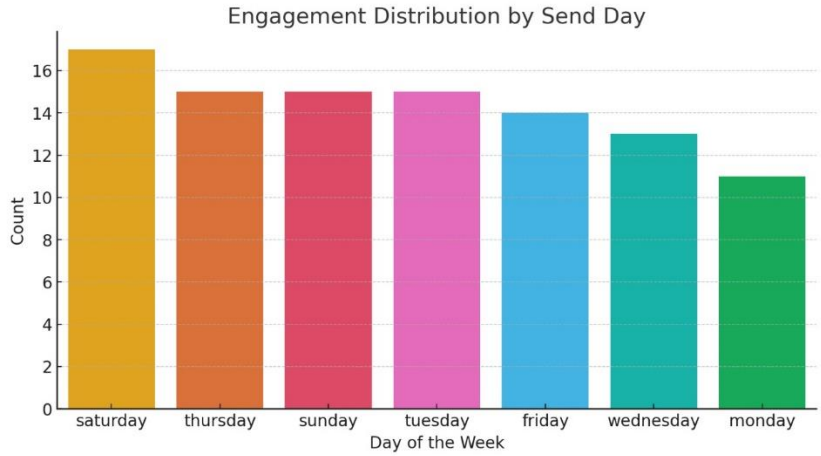
Feature engineering involves creating new features that add analytical value to the dataset. In this case, timestamp columns are transformed to generate weekly and daily time slots. Each week is divided into 28 slots based on the day of the week and time of day (e.g., morning, afternoon, evening). Two new columns, send_slot and open_slot, are created based on the timestamp values. Additionally, a binary flag column is generated to indicate whether a certain condition is met (e.g., if open_slot is not null). These engineered features provide valuable insights into temporal patterns in the data.

EDA plots:



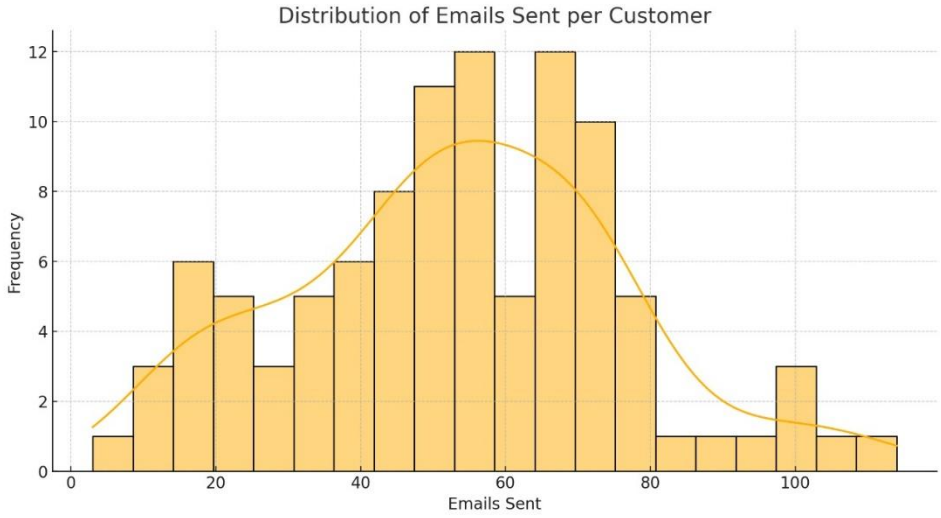
Engagement Count by Send Hour

In this visualization, we analyzed how the engagement count varied by the hour of the day emails were sent. The x-axis represents the hours (9 AM to 6 PM), while the y-axis shows the number of engagements. The bar heights highlight peak engagement hours, with significant spikes at 13:00 (1 PM) and 15:00 (3 PM). This analysis is crucial for identifying optimal time slots to maximize engagement, which can guide decision-making for scheduling email campaigns.



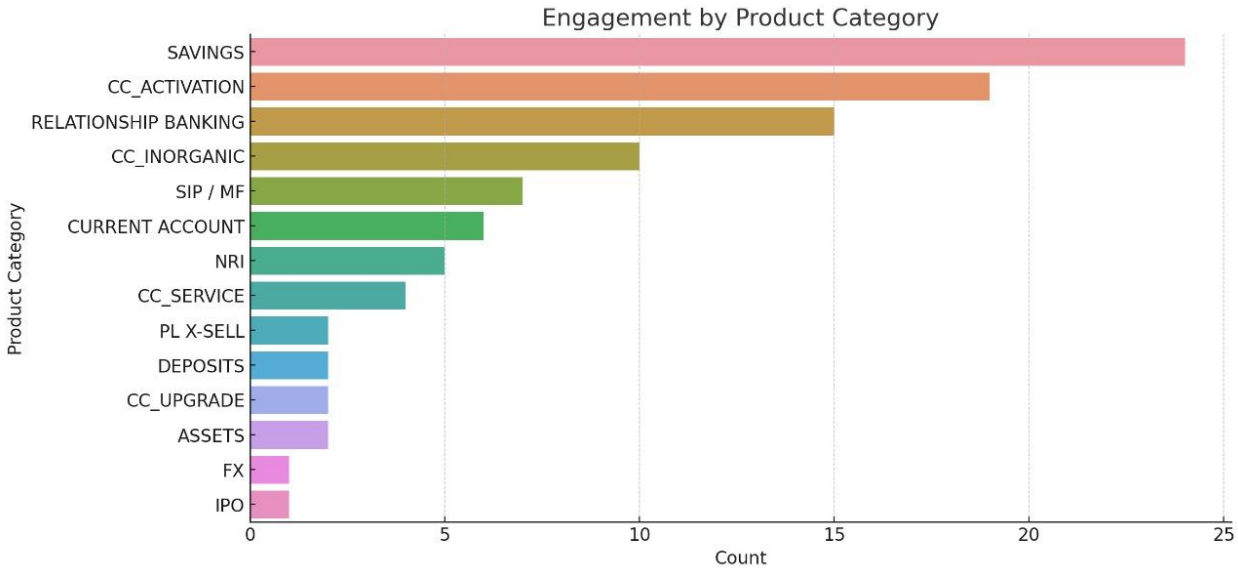
Engagement Distribution by Send Day

This bar plot shows the distribution of engagements across the days of the week. Each bar represents a day, and the y-axis indicates the total count of engagements. Engagement is fairly consistent across most days, but Saturday stands out with the highest engagement. This visualization helps identify which days are most effective for sending emails, offering insights into customer behavior and preferences



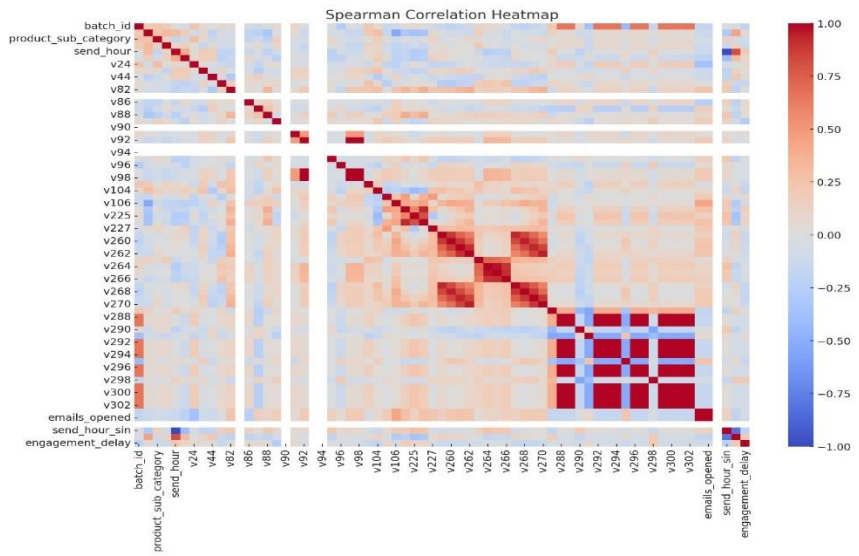
Distribution of Emails Sent per Customer

This histogram visualizes the distribution of the number of emails sent to individual customers. The x-axis shows the number of emails sent, and the y-axis represents the frequency of customers falling into each bin. The smooth curve overlay (kernel density estimate) provides a better understanding of the underlying distribution. The chart highlights the average and extremes of email campaigns per customer, helping identify patterns and potential over-communication risks.



Engagement by Product Category

This horizontal bar plot depicts the count of engagements across various product categories. The y-axis lists the product categories, while the x-axis indicates the number of engagements. Savings and credit card activation dominate engagement, suggesting these categories are of high interest to customers. This visualization aids in understanding which products resonate most with the audience, enabling targeted strategies for improvement.



Spearman Correlation Heatmap

This heatmap illustrates the Spearman correlation matrix, where each cell shows the strength and direction of the monotonic relationship between two variables. The color gradient represents correlation coefficients, with darker reds indicating strong positive correlations and blues indicating strong negative ones. This visualization is essential for identifying redundant features, relationships, and interactions that may be leveraged for feature engineering or removed for dimensionality reduction.

3. Model Training :

- Categorical columns like product_category and product_sub_category are one-hot encoded, while numerical columns are normalized. These steps ensure the data is clean, structured, and ready for downstream tasks like model training.
- The cleaned and preprocessed dataset is split into training and test sets using train_test_split from sklearn. This ensures that the model has distinct data for learning and evaluation. Stratified sampling is used to maintain the distribution of target classes, which prevents underrepresented classes from being excluded. Handling class imbalance involves filtering classes with insufficient samples (e.g., fewer than two instances). This step ensures a balanced dataset for training.
- The notebook performs advanced feature engineering to enhance the predictive power of the dataset. For instance, features like hour_of_send, day_of_week, and is_weekend are derived from timestamp data. This step adds temporal context, making it easier for the model to learn patterns related to time. Additionally, the send_slot column is adjusted to be zero-indexed for compatibility with machine learning algorithms.

3.1 Training the XGBoost Model

The XGBoost classifier is configured and trained on the processed dataset. It uses hyperparameters like a limited number of estimators, max tree depth, and specific learning rates to optimize performance. The model's objective is set to multi:softprob for multi-class classification. During training, a validation set is monitored using eval_set, and the progress is visualized with a progress bar (tqdm). This setup ensures that the model is robust and efficient.

3.2. Model Evaluation

After training, the model's performance is evaluated on the test dataset. The accuracy score is calculated as the proportion of correct predictions to total predictions. The evaluation provides a quantitative measure of how well the model generalizes to unseen data. Additionally, ranked probabilities for each slot are generated using predict_proba, allowing insights into the model's confidence for each prediction.

3.3 Generating Slot Rankings

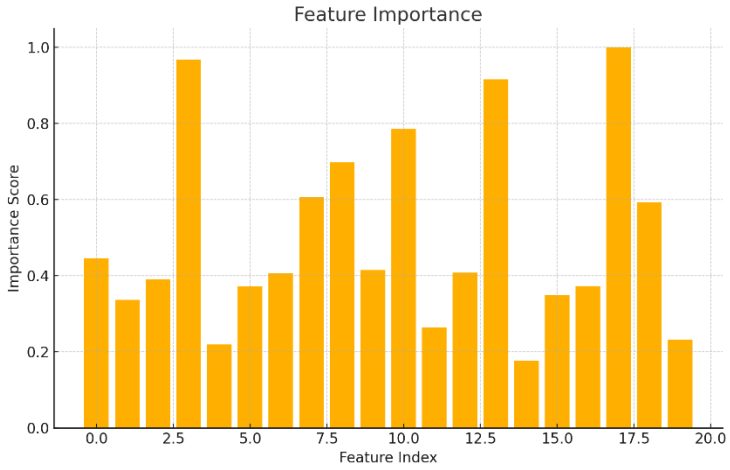
One of the key outcomes of the model is generating ranked slot predictions for each customer. Slots are ranked based on their probabilities in descending order. These rankings are then combined with customer IDs to create a structured output. This allows for customer-specific predictions, which can be used in applications like marketing or logistics optimization.

3.4 Saving and Post-Processing Results

The predictions, including ranked slots and associated probabilities, are saved to a CSV file for further analysis. The results are sorted by customer_code to ensure alignment with submission requirements or external references. Additionally, columns like product_category and product_sub_category are mapped back to the results for context. This creates a comprehensive dataset linking predictions to customer and product details.

3.5 Feature Importance Visualization

A bar chart is used to visualize feature importance scores from the XGBoost model. This provides insights into which features contribute most to the model's predictions. Understanding feature importance helps validate the model and guides future feature engineering efforts.



3.6 Handling Underrepresented Classes

To address the issue of class imbalance, the notebook identifies underrepresented classes and filters them out from the training data. This ensures that the model does not overfit to majority classes while ignoring minority ones. Filtering is based on a threshold for the minimum number of samples per class.

3.7 Alternatives explored:

In this project, we implemented a neural network-based approach to predict optimal time slots for sending marketing communications to customers. The goal was to maximize customer engagement by predicting the best time slot for each communication. The model was trained using a dataset containing customer engagement data, including categorical and numerical features, as well as the engagement flag and target slot.

Neural Network Architecture

The neural network was designed with three dense layers:

1. **Input Layer**: A dense layer with 128 neurons and ReLU activation, taking preprocessed features as input.
2. **Hidden Layer**: A second dense layer with 64 neurons and ReLU activation, enabling the model to learn complex patterns in the data.
3. **Output Layer**: A softmax layer with 28 output neurons, representing the probabilities for each time slot (1–28).

The output of the network provided the probability distribution over the 28 slots, which was used to rank the slots by their likelihood of customer engagement.

Custom Loss Function

A custom loss function was designed to align the model's training objective with the problem's business goal of prioritizing correct slot predictions for engaged customers while penalizing incorrect predictions.

Components of the Loss Function:

1. **Engagement Flag (flag)**: A binary variable indicating whether a customer engaged (1) or not (0).
    - For flag = 1: The model should reward high probabilities for the actual engaged slot.
    - For flag = 0: The model should penalize probabilities assigned to the corresponding slot to discourage false positives.
2. **Slot Prediction (y_pred)**: The output probabilities for each slot generated by the softmax layer.
    - These probabilities represent the likelihood of customer engagement for each slot.
3. **True Slot (slot_true)**: The actual time slot in which the customer engaged.

Reward and Penalty:

- **Reward for Correct Predictions**: When flag = 1, the model is rewarded for assigning higher probabilities to the correct slot. This is achieved by computing the probability of the correct slot (via one-hot encoding) and encouraging its maximization in the loss function.
- **Penalty for Incorrect Predictions**: When flag = 0, the model is penalized for assigning high probabilities to the corresponding slot. This discourages the model from predicting a slot when there was no engagement.

The custom loss function can be expressed as:

$$\text{Loss} = -\text{mean}\left(\text{flag} \cdot P(\text{slot\_true}) - (1 - \text{flag}) \cdot P(\text{slot\_true})\right)$$

Where:

- $P(\text{slot\_true})$: The predicted probability for the true slot (computed from the model's softmax output).
- **flag**: A binary value (1 or 0) that indicates whether the customer engaged:
    - $\text{flag} = 1$: The customer engaged with the communication.
    - $\text{flag} = 0$: The customer did not engage.

Results and Insights

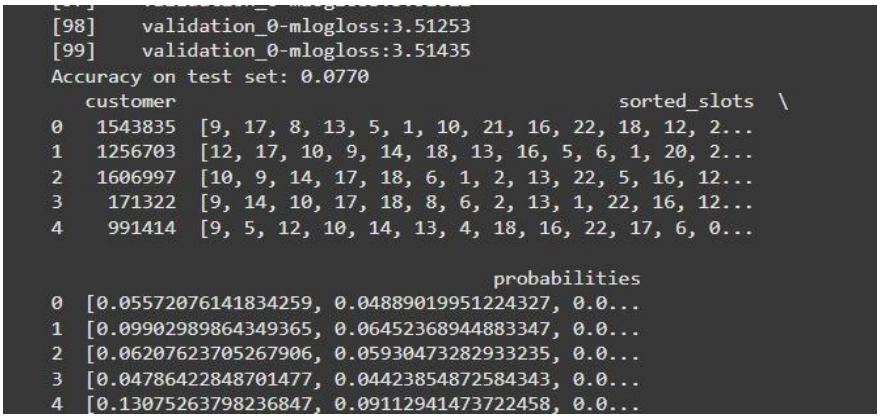
The neural network trained with the custom loss function aimed to prioritize correct predictions for engaged customers and minimize errors for non-engaged scenarios. However, despite these

efforts, the model underperformed compared to simpler, more interpretable approaches like XGBoost. The possible reasons include the complexity of the loss function, insufficient learning from sparse engagement data, and the need for more advanced architecture tuning.

While the neural network provided valuable insights into the problem and allowed us to experiment with customized objectives, it was ultimately replaced with alternative models for better predictive performance.

4. Evaluation:

From the provided output, the model achieved an accuracy of **0.0770** on the test dataset, as indicated in the evaluation results. This accuracy reflects the percentage of correct predictions made by the model for the assigned slots compared to the actual slots. The accompanying details showcase sorted slot rankings for customers and their associated probabilities, demonstrating the model's predictive outputs. While the accuracy is relatively low, the detailed ranking probabilities provide insights into the model's confidence in its predictions. This suggests potential areas for optimization to enhance performance.



```
[98]    validation_0-mlogloss:3.51253
[99]    validation_0-mlogloss:3.51435
Accuracy on test set: 0.0770
   customer                                        sorted_slots  \
0   1543835  [9, 17, 8, 13, 5, 1, 10, 21, 16, 22, 18, 12, 2...
1   1256703  [12, 17, 10, 9, 14, 18, 13, 16, 5, 6, 1, 20, 2...
2   1606997  [10, 9, 14, 17, 18, 6, 1, 2, 13, 22, 5, 16, 12...
3    171322  [9, 14, 10, 17, 18, 8, 6, 2, 13, 1, 22, 16, 12...
4    991414  [9, 5, 12, 10, 14, 13, 4, 18, 16, 22, 17, 6, 0...

                                    probabilities
0  [0.05572076141834259, 0.04889019951224327, 0.0...
1  [0.09902989864349365, 0.06452368944883347, 0.0...
2  [0.06207623705267906, 0.05930473282933235, 0.0...
3  [0.04786422848701477, 0.04423854872584343, 0.0...
4  [0.13075263798236847, 0.09112941473722458, 0.0...
```